Aniketh Girish
Amrita University • Amritapuri Campus • Amritapuri • Clappana P.O • Kerala • 690525
anikethgireesh@gmail.com

To introduce, I am Aniketh Girish, a final year student from Amrita University in India majoring in Computer Science. I am particularly interested in the intersection of network and systems as well as at the intersection of internet protocols and security.

Currently in my ongoing bachelor thesis - I'm evaluating the practical impact of QUIC for media streaming where the study is carried out in VLC media player. This study compare the Quality of Experience(QoE) of Real-time protocol (RTP) and Dynamic Adaptive Streaming over HTTP (DASH) over vanilla QUIC (draft-22). In the view of past studies, raw deployment of RTP and DASH over QUIC performs worse than TCP. Hence, we integrate a custom QUIC extension for streaming in VLC. The custom QUIC extension introduced aims at developing an optimised general-purpose QUIC extension for both streaming and web consisting of features such as Message abstraction, Forward Error Correction, selective reliable transport, head-of-line blocking resilience, deadline and dependency awareness and much more. Once the custom QUIC prototype is developed consisting of the above concepts, we deploy it in VLC and run a variety of performance tests to assess the QoE of streaming: (1) Comparison of RTP and DASH over vanilla QUIC (draft-22) based streaming stack in VLC. (2) Comparison of the RTP and DASH over the custom QUIC for the streaming stack which we developed as a part of this research.

A selected project in my field of interest would be when, I was selected as a Google Summer of Code[0] student for the 2nd time in the year 2018, where I was working on the project Wget2 under GNU Linux organisation. I was involved with the project to provide support for DNS over HTTPS (RFC 8484) in Wget2. DNS over HTTPS(DoH) is a web protocol that argues for sending DNS requests and receiving DNS responses via HTTPS connections, hence providing query confidentiality. Therefore to provide such a name resolution, I devised a library where I implemented the DNS protocol by facilitating the library to create the DNS packets - queried A, AAAA, CNAME records and implemented the DoH protocol by encoding the DNS wire format with base64 and sending it over HTTP/2. Further parsing the DOH response from the HTTP/2.0 response body to complete the resolution. Additionally, I am working on extending the DNS implementation with the integration of EDNS and added support for DNS-SD in wget2. With the experience and involvement in DoH, I'm involved and collaborating with IETF India chapter (IIESoc) to research and develop an RFC on 'Effects of DNS-over-HTTPS on Enterprise'.

Further, I am currently involved in remote research along with Prof. Taejoong (Tijay) Chung from Rochester Institute of Technology at the USA in measuring and finding security vulnerabilities of DANE protocol. The DANE (DNS-based Authentication of Named Entities) protocol takes advantage of the DNSSEC provided a chain of trust to authenticate TLS certificates. It places TLSA records in the DNS hierarchy and uses DNSSEC to validate their integrity. DANE TLSA, therefore, complements PKI Certificate Authorities, allowing TLS-users to better control certificates validation and protect against classes of CA compromise. The goal of the research is to measure/analyse the amount and quality of the deployment of DANE-related objects in .com, .net, .org, and other domains from a huge dataset that has been collected over a span of more-than-a-year. Next, the research dives into finding vulnerabilities/misconfigurations of mail servers deployed with DANE. DANE has just begun to be deployed at some mail servers to check the validity of the sender. Just like other PKIs (e.g., TLS, DNSSEC, and so on), there are some servers that would just accept the mail without properly checking the DANE certificate or just intake busted signatures.

Previous summer, I worked as a research associate at IIJ-II(IIJ Innovation Institute's Research Laboratory) in Tokyo, Japan on dockerd port to macOS advised by Dr Hajime Tazaki. Dockerd is the persistent process that manages containers which acts as the demon binary of docker. The dockerd port to macOS eventually started as a subset research goal carried in the redesign of container stack to experiment and measure extensible and platform independence attained in containers on `μKontainer` of which a library operating system was introduced - it decouples the kernel component and uses that as the container kernel within pure user space processes. On this light, docker has introduced a guest VM for docker to run on macOS - while docker in Linux run directly on the host machine. Therefore the port of docker relied on `μKontainer` to run docker directly on the macOS host. This resulted in an implementation of docker which runs independently of it's the host kernel. My work spanned across various processes within the entire docker runtime: a low-level runtime which manages the creation of such containers, **containerd** - the process which manages the containers, **dockerd** - the daemon incorporates building containers, managing the images, and running containers. The port of dockerd was focused on proper handling of the memory management API, preparation of root file system via graph drivers(filesystem manager) and network configurations.

Previously, I was selected for Google Summer of Code 2017 under KDE, where I worked on a project for a libre graphics software, Krita. My work involved introducing a data sharing module in it. The module enables communication between Krita and a remote KDE server in order to help users save and publish their data online. This also required modifying the underlying framework to enable client/server communication. The entire work and the code written were based on C++ and Qt.

Another experience which I would call close with a network (TCP/IP) stack and kernel development is where I'm developing a userspace network stack in Go called `Probe`. Usual protocol suite design tends to segregate each protocol differently and is handled as different processes and communication happen through these processes. While this TCP/IP stack segregates each packet/message handling as a separate process for reduced latency since this approach reduces context switching. Moreover, this stack introduces various data structures to implementation to carry message buffer for each protocol. For example, we emulate a packet buffer similar to the sk_buff kernel data structure. Further to simulate an independent network stack from the kernel network stack, a TAP device is introduced. A TAP interface is a virtual network interface, and it mimics actual hardware with simple software. The final binary produced could be used to run any sort of network-related functions, this binary replaces the kernel network stack to use our own userspace stack for network-related functionality.

If you find my application interesting, do take a look at the CV and drop a message and we can get the next level planned. Looking forward to hearing back from you soon.